Université Caen Normandie

U.F.R des Sciences
L2 Informatique

Jeu de Bataille Navale en Java

Rapport de projet du CC SINFL4A2

Réalisé par :
Akcel Arab
Belaid Azil
Massinissa Meghira
Mahdi Mobarek

Encadrant :
Ranaivoson Olivier

Année universitaire 2024-2025

Table des matières

1 Introduction

Ce rapport présente la conception et le développement d'un jeu de *bataille navale* en **Java**, réalisé dans le cadre du devoir de contrôle continu du module **SINFL4A2** à l'Université de Caen. Ce projet s'inscrit dans une approche pédagogique visant à renforcer notre maîtrise de la programmation orientée objet, tout en nous initiant à la conception d'une application logicielle complète, de l'architecture aux interfaces utilisateur.

L'objectif principal était de créer une application fidèle aux principes de l'architecture **MVC** (Modèle - Vue - Contrôleur), garantissant une séparation claire des responsabilités entre la logique métier, l'interface graphique et les interactions utilisateur. Cette organisation modulaire nous a permis de concevoir un projet structuré, maintenable et facilement extensible.

Le jeu développé propose deux modes d'interaction :

— Une **interface graphique (GUI)** intuitive, qui permet à l'utilisateur de jouer via des composants visuels,

— Et un **mode en ligne de commande**, offrant une alternative plus sobre, utile pour tester rapidement la logique du jeu ou jouer dans un environnement sans interface graphique.

Ce projet nous a également permis de mettre en œuvre des concepts essentiels tels que le *pattern Observateur*, utilisé pour permettre à la Vue d'écouter les changements du Modèle, ainsi qu'une gestion événementielle soignée pour synchroniser les différentes couches de l'application.

Dans les sections suivantes, nous présenterons d'abord l'architecture générale du projet, puis nous détaillerons la structure logicielle mise en place, avec une attention particulière portée au modèle MVC. Nous aborderons ensuite l'implémentation des mécanismes du jeu, avant de décrire les interfaces utilisateur, chacune pensée pour offrir une expérience adaptée au contexte d'utilisation. Enfin, nous conclurons avec un bilan du projet et les perspectives d'amélioration envisagées.

2 Architecture Générale du Projet

Nous avons structuré notre projet autour de **quatre packages principaux**, chacun jouant un rôle bien défini dans l'organisation globale de l'application. Cette séparation nous a permis de garantir une meilleure lisibilité du code, de faciliter sa maintenance, et de rendre l'application plus modulaire et évolutive.

On retrouve ainsi :

— un package dédié au **modèle**, qui contient la logique métier et les données du jeu,

— un package pour la **vue**, en charge de l'affichage graphique ou console,

— un package pour le **contrôleur**, qui gère les interactions entre l'utilisateur et le système,

— Un package pour les **listeners**, **listenables** et **notifications**, facilitant la communication entre les composants et permettant ainsi à la vue d'écouter le modèle, et non pas le contrôleur qui prévient la vue.

L'ensemble de cette organisation est illustré dans le diagramme suivant. Pour un aperçu plus détaillé, des diagrammes UML spécifiques à chaque package sont disponibles en annexe.

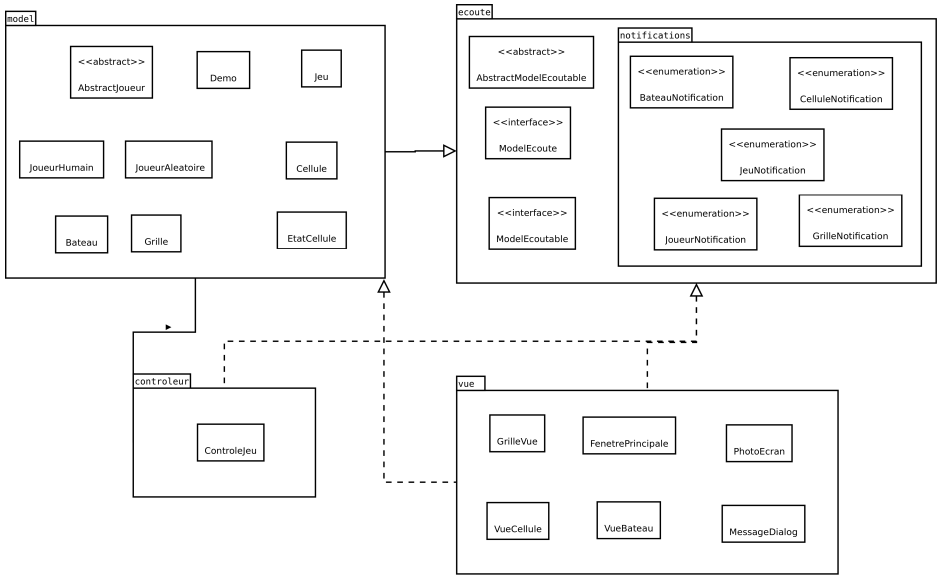

FIGURE 1 – Diagramme général de l'application

3 Architecture Logicielle

3.1 Modèle MVC

Dans le cadre de ce projet, nous avons choisi d'adopter l'architecture **MVC** (Modèle - Vue - Contrôleur), un pattern de conception largement utilisé pour séparer les différentes responsabilités d'une application. Ce choix nous a permis de structurer notre code de manière claire, cohérente et facilement maintenable.

- **Modèle** : C'est le cœur du jeu. Il gère l'état de la grille, les positions des bateaux, la logique des tirs, la détection des impacts et la gestion du déroulement des tours. Le modèle est entièrement indépendant de l'interface utilisateur, ce qui nous a permis de le tester facilement, notamment en mode console.
- **Vue** : Elle est chargée d'afficher l'état du jeu à l'utilisateur. Nous avons développé deux vues complémentaires : une version graphique à l'aide de la bibliothèque `Swing`, et une version en ligne de commande, utile pour les tests rapides ou pour jouer sans interface graphique. La Vue se met à jour automatiquement lorsqu'un changement survient dans le Modèle.
- **Contrôleur** : Il joue le rôle d'intermédiaire entre l'utilisateur et le Modèle. Il récupère les actions de l'utilisateur (clics, frappes clavier...), les interprète, et déclenche les mises à jour nécessaires du Modèle. Une fois le Modèle modifié, la Vue est notifiée et se rafraîchit.

Afin de permettre à la Vue de suivre les évolutions du Modèle sans créer de dépendance directe, nous avons mis en place un **package dédié** `ecoute`, qui centralise les interfaces d'observation. Ce mécanisme repose sur le *pattern Observateur*, qui nous a permis d'implémenter une communication unidirectionnelle efficace : la Vue s'inscrit comme observateur auprès du Modèle via ces interfaces, et elle est automatiquement informée dès que l'état du jeu change.

Grâce à cette architecture, la Vue reste totalement découplée du Modèle, ce qui respecte les bonnes pratiques du MVC et facilite l'évolution future de l'interface sans impacter le cœur de la logique métier.

4 Implémentation des Algorithmes

4.1 Gestion des tirs

Nous avons optimisé la vérification des tirs grâce à l'utilisation d'une matrice 2D, ce qui nous permet de localiser rapidement les impacts sans devoir parcourir l'ensemble de la flotte de manière linéaire, ce qui serait coûteux en temps.

Pour cela, nous avons mis en place une classe `Grille`, représentant une matrice définie par un nombre de lignes et de colonnes (`public Grille(int lignes, int colonnes)`). Cette structure nous offre un accès direct et rapide à chaque cellule du champ de bataille.

Dans la classe `AbstractJoueur`, la méthode `tire(int x, int y)` retourne un tableau contenant les coordonnées du tir. Ces coordonnées sont ensuite utilisées dans la méthode `tireGrilleAdversaire(int x, int y)` de la classe `Jeu`, qui vérifie l'état de la cellule ciblée dans la grille adverse. Si un bateau est touché, la cellule est mise à jour pour refléter l'impact (touché ou manqué).

Cette approche rend la gestion des tirs beaucoup plus efficace en réduisant le temps d'accès et de mise à jour des informations, tout en assurant une meilleure réactivité du jeu.

5 Interface Utilisateur

Nous avons respecté ce qui est demandé dans l'énoncé du devoir, c'est pourquoi nous avons implémenté les deux possibilités pour jouer : un mode ligne de commande et un mode graphique.

5.1 Mode Ligne de Commande

Le jeu peut être joué directement dans la ligne de commande, avec une représentation ASCII de la grille. Cette interface permet au joueur de suivre l'état du jeu de manière textuelle, avec une visualisation claire des tirs effectués. Chaque mouvement est reflété directement dans la ligne de commande. Comme illustré dans la Figure 2, la grille est affichée avec les positions des tirs et des vaisseaux.

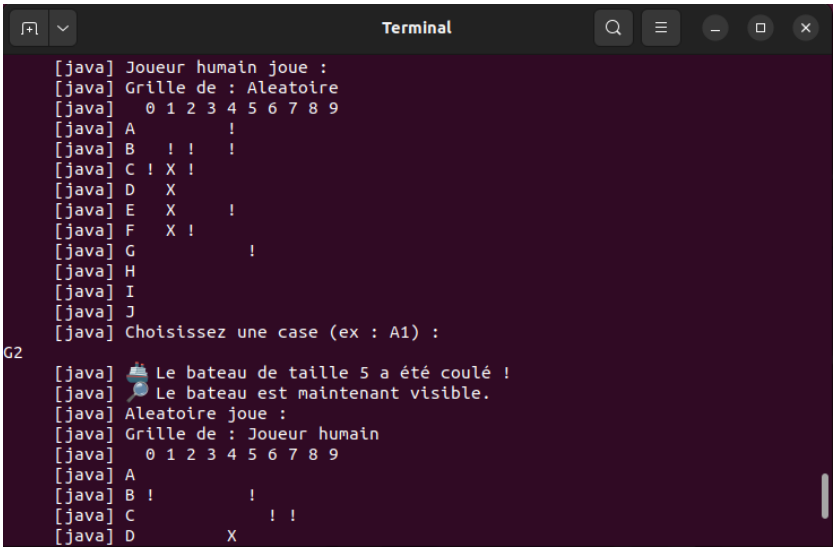

FIGURE 2 – Représentation ASCII du jeu en mode ligne de commande.

5.2 Mode Graphique

En mode graphique, une interface Swing permet au joueur d'interagir avec une grille interactive en cliquant pour tirer. Les tirs réussis sont affichés en rouge, tandis que les tirs manqués apparaissent en vert. Cette interface améliore l'expérience utilisateur en offrant une interface visuelle claire et interactive. Comme illustré dans la Figure 3, la grille interactive et les retours visuels sont visibles pendant le jeu.

FIGURE 3 – Interface graphique du jeu avec la grille interactive.

6 Conclusion

Ce projet nous a permis de mettre en pratique les principes de conception, en particulier l'architecture MVC et la séparation des responsabilités. Nous avons également optimisé certains algorithmes pour rendre le jeu plus efficace et agréable à jouer. Cependant, il reste encore des pistes d'amélioration intéressantes pour l'avenir. Parmi celles-ci, on pourrait envisager :

— Le développement d'une intelligence artificielle plus sophistiquée. Par exemple, lorsqu'un tir touche une cellule d'un bateau (une case rouge), la machine pourrait alors concentrer ses attaques sur les cases avoisinantes, augmentant ainsi ses chances de couler le bateau.

— L'ajout d'un mode multijoueur en réseau pour permettre aux joueurs de s'affronter en ligne.

7 Annexes

7.1 Diagrammes UML détaillés

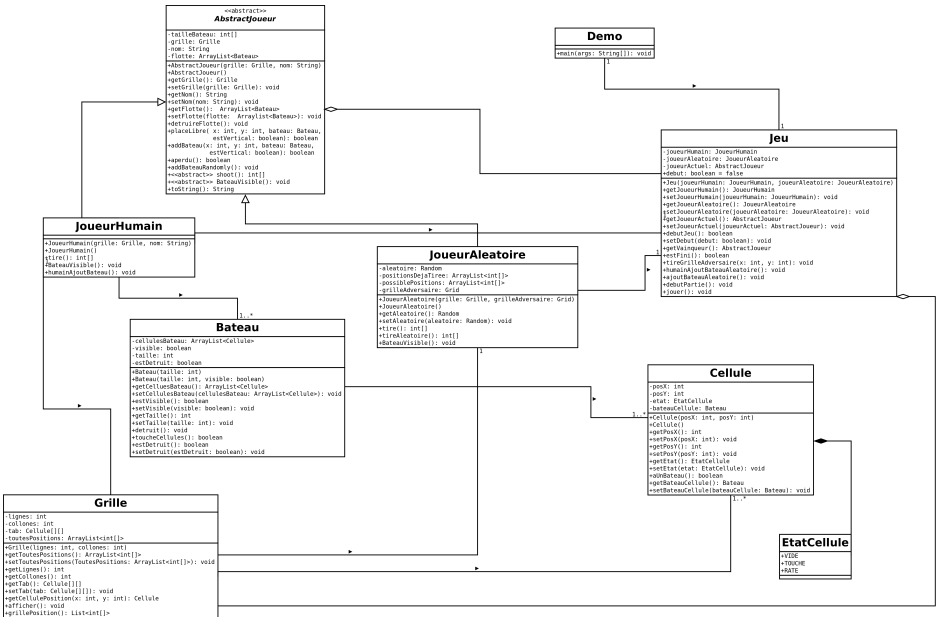

FIGURE 4 – Diagramme du modèle (package model)

FIGURE 5 – Diagramme de la vue (package vue)

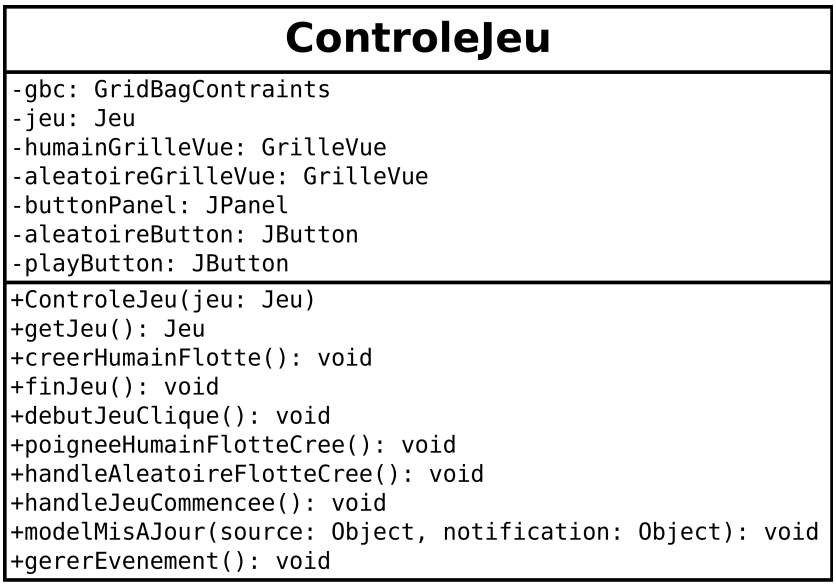

FIGURE 6 – Diagramme du contrôleur (package controleur)

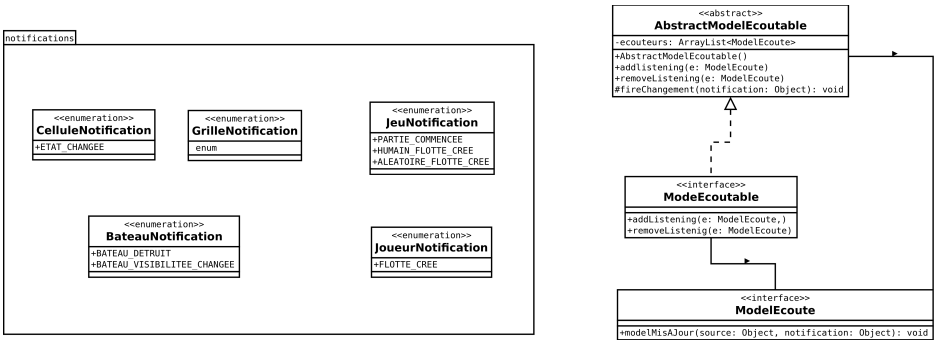

FIGURE 7 – Diagramme des écouteurs (package ecoute)